Synthesis of Sound Effects for Computer Games

J. R. Parker and Brad Behm

University of Calgary
Digital Media Laboratory
jparker@ucalgary.ca

***Abstract*—In the computer game context, it could be a significant advantage to be able to synthesize sounds given one or two relatively small examples. Not only do audio files, especially good quality ones, occupy a significant amount of storage space in memory or on a CD, but the flexibility of sound creation on demand and the increased quality and reduced repetition affect the game play experience positively. We will discuss a method for the synthesis of sound effects, and briefly summaries a method for synthesis of sound textures.**

***Index Terms*—sound synthesis, audio, computer games, audio display, sound effects.**

I. Introduction

TWO years ago we created a set of methods for the synthesis of audio from examples for a class of sounds we called *audio textures*. An audio texture can be described as having a somewhat random character, but a recognizable quality. Any small sample of a sound texture should sound very much like, but not identical to, any other small sample. We devised three methods for texture synthesis, but it is the one based on quilting that we used in this work. The idea behind this method is to cut pieces from the audio samples and stitch them together in a new order. Of course, pieces can be reused, in whole or in part, allowing much bigger images to be built from smaller samples. There are various ways to deal with the fact that the seams between the samples are often audible.

The texture methods do not work at all well on short sounds of the type we label sound effects: gunshots, missile launches, impact sounds, and so on. The reason is that these types of sounds consist of a beginning, a middle, and an end, while a texture is really a continuous statistically consistent pattern that does not start or stop. In order to generate a sound effect with fidelity a connection must be established between the temporal relationship found in the sound and the creation of the textures. What is the frequency and intensity variation in a a gunshot sound, or a car crash? How does a breaking glass change in frequency as a function of time? As a result of these issues that are temporally based variables, we began a study of sound effects as opposed to the textures we had examined until then.

We started by looking at amplitude-time plots of sounds taken from games, searching specifically for an envelope that modulates the amplitude for these sounds. This may be key to their realistic synthesis. The envelope is a function that characterizes the volume changes in a sound sample. We decided to create an envelope by creating an amalgam of many effects in the same class (I.E. gunshots) from input files. They are summed into a buffer and broken into 11 regions. Then line segments are drawn between the enclosing points, generating a function that looks something like the curve seen in the figure; this particular envelope was created from a set of missile-launch inputs. This scheme worked only fairly well, as can be heard using the samples on our web page (http://pages.cpsc.ucalgary.ca/~behmb/audio/). Following this, two post processing steps were used: smoothing with a 5 sample wide median filter, followed by volume (intensity) normalization.

II. Sound Textures: Tiling and Stitching

Audio synthesis can be done in a few different ways, and we have experimented with three schemes [4] based on what we call *particle audio*. The basic idea is to chop up a digitized audio signal

into small sequences, each the same duration between 0.001 and 0.1 seconds each. These are called *particles*, and have a superficial resemblance to pixels in computer graphics. However, where a pixel is really an average color or intensity over a small spatial region, an audio particle is a collection of samples, each of which is a temporal average or maximum, where their order is unchanged from the original sound sample. There are as many as 3000 samples in a particle, and relationships between particles are as important as relationships of samples within particles.

This idea has a resemblance to the idea of *granular synthesis* found in modern music composition [6,7]. In granular synthesis, a particle is called a *granule*, and is usually specified as a function having a frequency or set of frequencies, and envelopes, and other synthesizer type specifiers. Notwithstanding this, Xenakis used to slice magnetic tape into small bits and glue them together in a different order to create a composition, an act not unlike what we will soon propose. Figure 1 shows a general picture of a granule, as proposed by these pioneers.

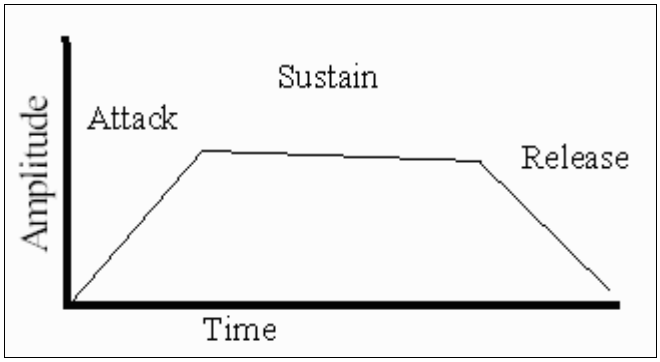


Figure 1 – The envelope enclosing a sound, showing the attack, sustain, and release phases.

Gabor [3] of *Gabor transform* fame had the idea that sound was actually perceived as a series of short bursts of energy about 10-21 milliseconds long. This later proved to be true, and is partly what allows digital reproduction of sound on computers, the Internet, and on CDs and DVDs.

A. *Lessons from Computer Graphics - Textures*

There has been some relatively recent work on the synthesis of graphical textures from pre-existing examples. The idea is simple enough: given an existing visual texture pattern such as foliage, brick, grass, and others, extend the pattern to fill in a larger area. This may be needed if an image is placed into a larger frame and the boundaries need to be extended, or if a small pan is wanted to give the illusion of motion and there is no more image to pan over.

A straightforward and easy to understand method for constructing new visual textures from smaller ones is to cut pieces from the samples and paste them together in a new order. Of course, the original image can be cut apart in many different ways and pieces can be reused multiple times, allowing much bigger images to be built from smaller samples. There are many ways to deal with the fact that the seams between the samples are often visible. This process is referred to as *image quilting* by Efros [2]. He describes a method that uses square sample blocks of a fixed size, having an overlap between neighboring blocks in all directions. Instead of selecting random blocks to be adjacent in the new image, it is profitable to select blocks that have some significant measure of agreement between them in the overlap region. Smoothing the edges reduces the visibility of the joins between the blocks but does not eliminate it.

B. *Audio Texture Synthesis*

The MIT Media Laboratory [5] has developed one of the very few methods for the synthesis of audio textures. They describe a two-level representation of sound, where the low level consists of what they call *atomic elements*, and the high level is a description of the distribution of those elements.Their implementation used a binary tree structured filter to extract the atoms, and at the high level the probability of each atom is estimated based on the previous ones and a statistical model is constructed. They used this scheme to generate some simple sounds: two sine waves, photocopier noise, and applause, but not without difficulties: clicks and pops appear when not expected, and some periodicity and flavor (*similarity*) was lost.

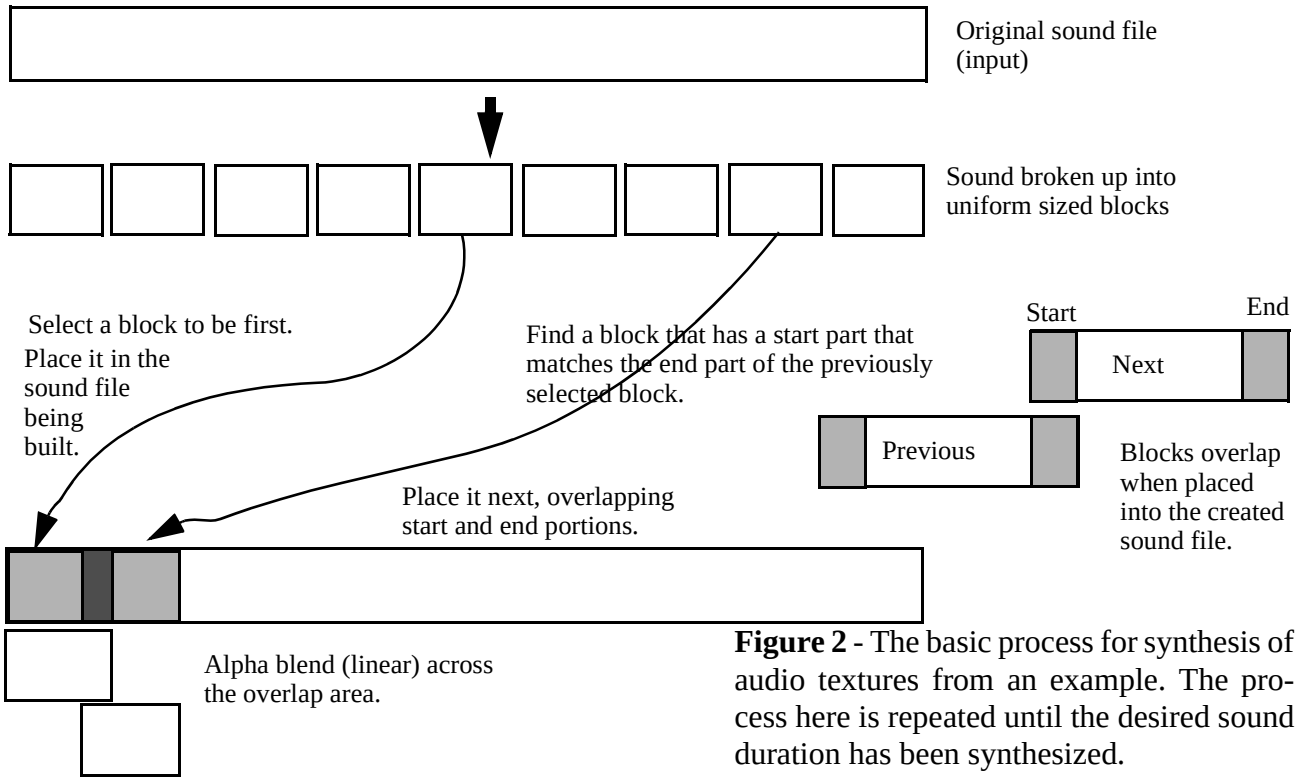


Figure 2 - The basic process for synthesis of audio textures from an example. The process here is repeated until the desired sound duration has been synthesized.

C.The New Algorithm

As a first step towards the generation of a new audio texture, rearranging and concatenating small samples of a source texture sound was tried, in a way that recalls the image quilting method for visual textures. The simplest way to do this is to repeatedly copy chunks of data from random points within the source into a new audio texture. It is important to see that by using the original sound cut into parts, we keep the essential sound quality of the original texture while introducing variety by altering the order of the pieces (particles). When we do this naively, clearly audible defects are heard in the transitions between random chunks. Obviously, a method of smoothing these transitions is necessary. We use a cross-fade (*Alpha blending*) algorithm that blends the tail of the preceding chunk into the head of the new chunk. The length of the blended portion is variable, but we found that 15% of the chunk size gives subjectively good results. The cross-fade between samples helps minimize the distortion in the transition between random audio chunks, but does not eliminate it. Portions of the overlap are audible for a significant part of the transition, and when collected into 10-20 transitions per second they result in clicking and buzzing sounds. Figure 2 illustrates the clipping, joining, and fading scheme described up to now.

Since an audio texture is, by definition, slightly repetitive, it is not necessary to rely on randomly choosing the next chunk in the texture. There should always be some number of chunks that could logically follow the previous chunk. That is, given one chunk or section of a sound, the next section in sequence would be the obvious choice for a next particle, but that would cause a repetition that might be recognizable. The fact that there is an obvious, preferential sequence is because the tail of the particle we just inserted has a high degree of similarity to the first part of the next one, and so on. The original sequence in may well be the best in this sense, but there are many others that are almost as good. Taking advantage of this eliminates distortion along transitions and enforces the similarity between the source and final texture.

We use a least-squares similarity measure to find

the regions in the source sample that are similar to the tail (that is, the last 15%) of the preceding chunk.

Since the chunks originate from the source sample, there is going to be one near perfect match, exactly where the tail occurs in the source. Allowing the copy to continue from this point results in the original sample simply being copied repetitively into the final texture, effectively tiling the source. This is undesired in a texture, so we forbid this behavior, and mark used sections as unusable for the immediate future.

This method of choosing chunks provides nearly seamless transitions, but also creates some obvious tiling or repetition in the long term, which we think of to be a defect in a sound texture. It is common for this algorithm to get stuck in a loop, in which it returns to a specific spot after a fixed number of chunks. A method is needed to encourage the chunk-selection algorithm to make use of the entire source sample, and not get stuck using the same regions over and over. This is done using an accumulator vector of the same size as the source sample.

At initialization, this accumulator is set to zero. When samples are copied out of the source, the corresponding bins in the accumulator are incremented by 5. After each iteration of the copy phase, all non-zero elements in the accumulator are decremented by one. When searching for the best match using the least-squares similarity measure, as above, only spaces whose accumulator 'rank' is zero are searched. If there are not enough consecutive zero-ranked elements to form the head of the new chunk, then the accumulator is decremented and the search is run again. This is very similar to the *least recently used* (LRU) algorithm used in virtual memory page replacement.

This technique forces the search algorithm to use all of the available material in the source audio file. This sometimes results in less-than ideal matches, but we have found the resulting transitions to be unnoticeable. It is important to note that when searching for a suitable chunk, only the rank of the lead-in segment (first 15%) is used. For this reason it is still possible for frequently used segments to be used several times, but their increasing rank will eventually cause the algorithm to look elsewhere. It is this behavior that prevents segments from becoming available in the same order that they were used, and thus reduces tiling and repetition.

The size of the chunks used in all of the above methods determines to a great extent the quality of the texture. The optimal chunk size depends on the specific source sample used. Less 'busy' samples, with a lower incidence of audible features, need longer chunks to avoid choppiness in the final result. Busier samples require smaller chunks to avoid audible tiling. By hand tailoring this size to a specific source sample, very good results can be obtained, but at the expense of generality.

One way to automatically determine the size of the chunks is using amplitude peaks. The entire source sample is analyzed for RMS amplitude, and peaks in amplitude more than 1.5 standard deviations from the baseline are recorded. The mean and standard deviation of the observed distance between these peaks is used to generate the size of each chunk. Hopefully, then, each chunk will contain one 'feature' that a listener can recognize. This method works reasonably well on most textures, but is still somewhat experimental. In the future, we plan to experiment with several frequency-based search strategies to determine the optimal chunk size. It is possible, however, that there is no method that will correctly determine optimal chunk size for all inputs.

III. Sound Effects

A sound effect is not a sound texture, for many reasons. The main reason is that an effect usually has a start, middle, and end part, very much related to the attack, sustain, release portions of a granule. A texture, one the other hand, has much the same statistical properties anywhere on the temporal axis. The process for creating a sound effect from s set of examples is the same as that for creating a sound texture, with two main additions: the creation of an envelope, and the use of local sampling.

A. *Creating an Envelope*

Given that there may be multiple input samples, the problem of creating an envelope to be used in synthesis is more or less complex depending on the samples themselves. It would be unusual to find that all of the samples would have the same duration, so the first step is to stretch or compress them so that they do. This is essentially a resampling step. The magnitudes are rescaled at the same time so that all of the samples have the same basic properties. A new, single data image is created that is the average (mean) of all of the normalized sound samples. This is the *amalgam* image.

The envelope is the boundary of the signal region in the resulting amalgam image, as shown in Figure 3. This amalgam is then broken into 11 regions and the root-mean-square (RMS) amplitude of the region is calculated. Then line segments are drawn between the enclosing points, generating a function which looks something like the curve seen in Figure 3b - this particular envelope was created from a set of missile-launch sound effect inputs. Graphing the individual inputs confirms that the envelopes are reasonable approximations of the amalgamation of the inputs.

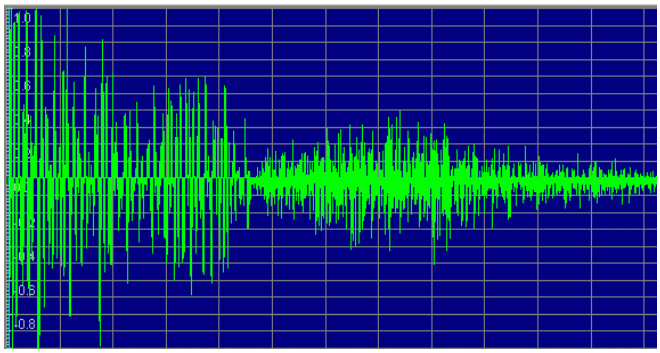
The envelope created in this manner is used to define constraints on the generated audio. The idea is to use the envelope as a region to fill, and to avoid exceeding the boundary. Low magnitude regions will be stretch to fill the envelope if they are too far from it.

B. *Use of Locality*

Locality in the current context is a reference to operating systems, virtual memory in particular. If a page in a virtual memory system is accessed at time T, it is more likely to be accessed again by time T+D than in an identical time interval in the more distant future.

In the creation of synthetic sound effects, a standard tiling and stitching method as described in section II was used, but when searching for the next sample to use, the search was restricted to an area (time interval) near the current position in the output stream. We decided to restrict the size of the window to 15% of the size of the input. One variant of this algorithm uses the absolute position in the output stream to identify where to search the inputs for matches. For example, suppose we have just finished pasting **M** samples into the output stream, making it a total of **N** samples long. We will search for matches in each input within +/- 7.5% of the index **N**. If an input sample has a total length less than **N** samples, it will not be considered in the synthesis of the selected sound.

A second variant, called *relative positioning*, uses a progress ratio. If the output stream is supposed to be Lout samples long, and we have written out **N**,



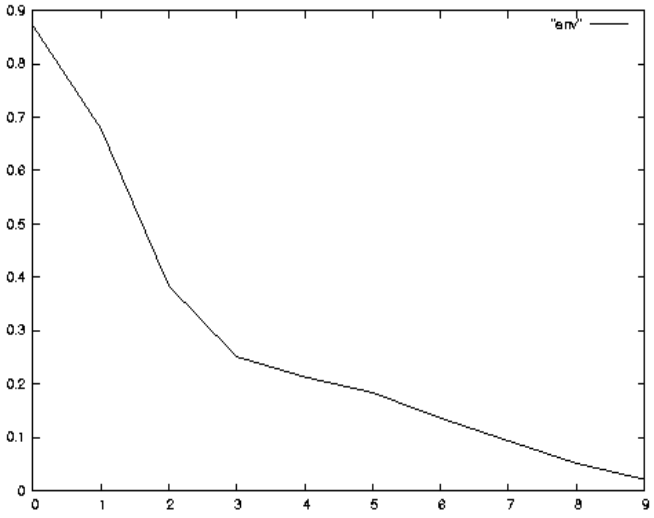


Figure 3. (a) Above, a sound effect to be synthesized. We need to find the envelope. (b) Right, the envelope found by using a linear boundary of 11 line segments.

we will search the region within +/- 7.5% of (Lout/N)*Lin.

Relative positioning seems to be consistent, and doesn't have the noticeable artifacts of the absolute method, but it is also a less dramatic improvement over the non-location-sensitive method. Examples are also available on the web page.

C. Constricting Minimum Window Size

For certain data sets, noticeably the engine sounds, there is a surfeit of extra high-frequency noise. This is caused by small artifacts at the places where the blocks meet adding to create a sound having a frequency related to the number of blocks per second. In an attempt to reduce this, the minimum size of the paste window was constrained to be 100 samples. Of course, ideally this should be sample-rate independent.

IV. Conclusions

The method described here can create synthetic sound effects from single or multiple input samples. If the samples represent the same sound, and have an observed similarity in timbre, pitch, and temporal properties, then the resulting synthetic sounds have been judged by a small group of listeners to be very similar to the originals. The method was designed for use with computer games, and the synthesis can be performed in real time as needed.

You may judge the quality by visiting the web page at http://pages.cpsc.ucalgary.ca/~parker/audio/ . This page is organize temporally, so the best results are at the bottom of the page.

Acknowledgments

The authors thank Sonny Chan for help with the first versions of the software.

References

[1] Z. Bar-Joseph, D. Lischinski, and M. Werman, Granular Synthesis of Sound Textures Using Statistical Learning, *Proceedings ICMC*, 1999.

[2] A. Efros and W.T. Freeman, Image Quilting for Texture Synthesis and Transfer, *Proc. SIGGRAPH 2001*, Los Angeles, Aug. 12-17, 2001

[3] Gabor, D. Acoustical quanta and the theory of hearing. *Nature*, 159(4044), 591-594, 1947.

[4] Parker, S. Chan, Sound Synthesis for the Web, Games and Virtual Reality, *SIGGRAPH 2003*, San Diego, CA. July 28-30, 2003.

[5] N. N. Saint-Arnaud and K. Popat, Analysis and Synthesis of Sound Textures, *Proc. IJCAI-95 Workshop on Computational Auditory Scene Analysis*, Montreal, August 1995, Pp. 125-131.

[6] Truax, B., Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2), 14-26, 1988

[7] Xenakis, Iannis, Formalized Music. Bloomington: Indiana University Press. 1971

J.R. Parker (B.Sc. Applied Mathematics, Calgary 1977; MSc. Computer Science, Calgary, 1980; Ph.D. Informatics, Gent, 1998) has been a member of the IEEE irregularly since 1988.

He is the author of three books, including the very popular *Algorithms for Image Processing and Computer Vision*.(1997) and *Start Your Engines: Developing Racing and Driving Games* (2005). He is a professor of Computer Science at the University of Calgary, where he specializes in machine perception, computer games, and artificial intelligence.

Dr. Parker is also a member of the Association for Computing Machinery, and the International Game Developers Association.